本节内容

线性表的逻辑结构

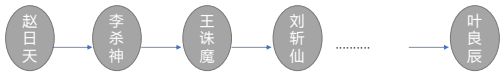
线性表

定义：线性表是具有相同数据类型的n（n≥0）个数据元素的有限序列。其中n为表长。
当n=0时 线性表是一个空表

结点

线性表

食堂里同学排队



学生信息表

学生信息表

学生姓名	性别	年龄	出生时间
张三	男	1	20110623
李四	女	2	20100106
王五	女	3	20090830
郑六	男	4	20080615
孙七	女	5	20070521
赵八	男	6	20060416

线性表中第一个元素称为表头元素；最后一个元素称为表尾元素。

除第一个元素外，每个元素有且仅有一个直接前驱。除最后一个元素外，每个元素有且仅有一个直接后继。

本节内容

线性表的顺序存储结构

顺序存储

图书馆占座

面包切片

顺序存储

线性表的顺序存储又称为顺序表。

它是用一组地址连续的存储单元（比如C语言里面的数组），依次存储线性表中的数据元素，从而使得逻辑上相邻的两个元素在物理位置上也相邻。

数组下标	顺序表	内存地址
0	a_1	LOC(A)
1	a_2	LOC(A)+sizeof (ElemType)
	⋮	
i−1	a_i	LOC(A)+(i−1)×sizeof (ElemType)
	⋮	
n−1	a_n	LOC(A)+(n−1)×sizeof (ElemType)
	⋮	
MaxSize−1	⋮	LOC(A)+(MaxSize−1)×sizeof (ElemType)

注意：线性表中元素的位序是从1开始的，而数组中元素的下标是从0开始的 在具体的题目中需要分辨清楚

顺序表任意元素可以在单位时间内找到存储位置

顺序存储

如何建立顺序表的结构呢？

首先我们要在内存中"找块地"，而且是连续的，那么我们可以先确定存储空间的起始位置；然后还要知道这块地有多大，那么这块地的大小我们也要确定；最后我们要将表中各个元素对号入座，那就要知道有多少元素，也就是表的长度。

建立顺序表的三个属性：1.存储空间的起始位置（数组名data）
2.顺序表最大存储容量（MaxSize）
3.顺序表当前的长度（length）

PS：宏定义

```
#define MaxSize 50                  //定义线性表的最大长度
typedef int Elemtype                //假定表中元素类型是int
typedef struct{
        ElemType data [MaxSize];    //顺序表的元素(数组)
        int length;                 //顺序表的当前长度
}SqList;                            //顺序表的类型定义
```


顺序存储

这里线性表的数组data是静态分配（开数组）的，大小固定,一旦满了就溢出

其实数组还可以动态分配空间，存储数组的空间是在程序执行过程中通过动态存储分配语句分配

这里不再开数组辣

```
typedef int Elemtype;
typedef struct{
        ElemType *data;                 //指示动态分配数组的指针
        int MaxSize,length;             //数组的最大容量和当前个数
} SeqList;
```

C语言的动态分配语句为

```
#define InitSize 100

SeqList L;

L.data=(ElemType*)malloc(sizeof(ElemType)*InitSize);
```

注意：动态分配并不是链式存储，同样还是属于顺序存储结构，只是分配的空间大小可以在运行时决定。

顺序存储

总结：

1.顺序表最主要的特点是随机访问（C语言中基于数组），即通过首地址和元素序号可以在O(1)的时间内找到指定的元素。

2.顺序表的存储密度高，每个结点只存储数据元素。无需给表中元素花费空间建立它们之间的逻辑关系（因为物理位置相邻特性决定）

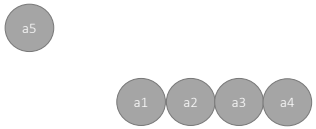
3.顺序表逻辑上相邻的元素物理上也相邻，所以插入和删除操作需要移动大量元素。

本节内容

顺序表的操作

顺序存储

1.插入



在顺序表L的第i（1≤i≤L.length+1）个位置插入新元素e。如果i的输入不合法，则返回false，表示插入失败；否则，将顺序表的第i个元素以及其后的所有元素右移一个位置，腾出一个空位置插入新元素e，顺序表长度增加1，插入成功，返回true。

顺序存储

算法思路：

1.判断i的值是否正确
2.判断表长是否超过数组长度
3.从后向前到第i个位置，分别将这些元素都向后移动一位
4.将该元素插入位置i 并修改表长

区别位序和数组的下标

PS：bool 是布尔类型它只有true和false两种值，分别表示真和假

```
bool ListInsert(SqList &L , int i , ElemType e){
        if(i<1||i>L.length+1)                    //判断i的范围是否有效
                return  false;
        if(L.length>=MaxSize)                    //当前存储空间已满，不能插入
                return  false;
        for(int j=L.length ; j>=i; j--)          //将第i个元素及之后的元素后移
                L.data[j]=L.data[j-1];
        L.data[i-1]=e;                           //在位置i处放入e
        L.length++;                              //线性表长度加1
        return  true;
}
```

顺序存储

分析顺序存储的插入算法

```
bool ListInsert(SqList &L , int i ,  ElemType e){
        if(i<1||i>L.length+1)
                return  false;
        if(L.length>=MaxSize)
                return  false;
        for(int j=L.length ; j>=i; j--)
                L.data[j]=L.data[j-1];
        L.data[i-1]=e;
        L.length++;
        return  true;
}
```

最好情况：在表尾插入（即i=n+1），元素后移语句将不执行，时间复杂度为O(1)。
最坏情况：在表头插入（即i=1），元素后移语句将执行n次，时间复杂度为O(n)。
平均情况：假设pi（pi=1/(n+1)）是在第i个位置上插入一个结点的概率，则在长度为n的线性表中插入一个结点时所需移动结点的平均次数为

$$\sum_{i=1}^{n+1} p_i(n-i+1)=\sum_{i=1}^{n+1}\frac{1}{n+1}(n-i+1)=\frac{1}{n+1}\sum_{i=1}^{n+1}(n-i+1)=\frac{1}{n+1}\cdot\frac{n(n+1)}{2}=\frac{n}{2}$$

线性表插入算法的平均时间复杂度为O(n)

顺序存储

2.删除

删除顺序表L中第i（1≤i≤L.length）个位置的元素，成功则返回true，并将被删除的元素用引用变量e返回，否则返回false。

顺序存储

算法思路：

1.判断i的值是否正确
2.取删除的元素
3.将被删元素后面的所有元素都依次向前移动一位
4.修改表长

```
bool ListDelete(SqList &L, int i, Elemtype &e){
      if(i<1||i>L.length)                    //判断i的范围是否有效
            return  false;
      e=L.data[i-1];                                  //将被删除的元素赋值给e
      for(int j=i;j<L.length;j++)            //将第i个位置之后的元素前移
            L.data[j-1]=L.data[j];
      L.length--;                                     //线性表长度减1
      return  true;
}
```


顺序存储

```
bool ListDelete(SqList &L, int i, Elemtype &e){
      if(i<1||i>L.length)
            return  false;
      e=L.data[i-1];
      for(int j=i;j<L.length;j++)
            L.data[j-1]=L.data[j];
      L.length--;
      return  true;
}
```

最好情况：删除表尾元素（即i=n），无须移动元素，时间复杂度为O(1)。
最坏情况：删除表头元素（即i=1），需要移动除第一个元素外的所有元素，时间复杂度为O(n)。
平均情况：假设pi(pi=1/n)是删除第i个位置上结点的概率，则在长度为n的线性表中删除一个结点时所需移动结点的平均次数为

$$\sum_{i=1}^{n} p_i(n-i)=\sum_{i=1}^{n}\frac{1}{n}(n-i)=\frac{1}{n}\sum_{i=1}^{n}(n-i)=\frac{1}{n}\cdot\frac{n(n-1)}{2}=\frac{n-1}{2}$$

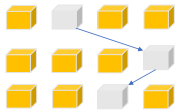
线性表删除算法的平均时间复杂度为O(n)

本节内容

线性表的链式存储结构

链式存储

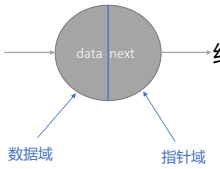
顺序结构需要一块连续的存储空间，那如果我们只有零散的空间呢？



线性表的链式存储是指通过一组任意的存储单元来存储线性表中的数据元素。

为了建立起数据元素之间的线性关系，对每个链表结点，除了存放元素自身的信息之外，还需要存放一个指向其后继的指针。

链式存储

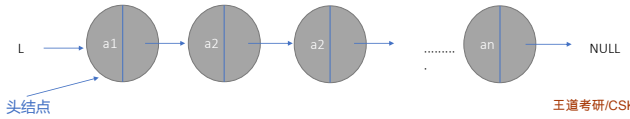



```
typedef struct LNode{        //定义单链表结点类型
   ElemType data;            //数据域
   struct LNode *next;       //指针域
}LNode, *LinkList;
```

因为每个结点只有一个指针指向下一个结点，故又称单链表

通常用“头指针”来标识一个单链表， 例如Linklist L 那么头指针L就代指一个单链表，头指针为“NULL”时则表示一个空表。

单链表第一个结点之前附加一个结点，称为头结点。头结点的数据域可以不设任何信息，也可以记录表长等相关信息。头结点的指针域指向线性表的第一个元素结点



链式存储

头结点和头指针的区别？

不管带不带头结点，头指针始终指向链表的第一个结点，而头结点是带头结点链表中的第一个结点，结点内通常不存储信息

为什么要设置头结点？

1.处理操作起来方便 例如：对在第一元素结点前插入结点和删除第一结点起操作与其它结点的操作就统一了
2.无论链表是否为空，其头指针是指向头结点的非空指针，因此空表和非空表的处理也就统一了。

本节内容

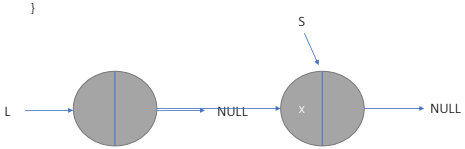
单链表的操作(一)

单链表的操作

1.头插法建立单链表：

建立新的结点分配内存空间，将新结点插入到当前链表的表头


```
LinkList CreatList1(LinkList &L){
        LNode *s;              //辅助指针
      int x;
        L=(LinkList)malloc(sizeof(LNode));          //创建头结点
        L->next=NULL;                                //初始为空链表
        scanf("%d",&x);                                          //输入结点的值
        while(x!=9999){                                          //输入9999表示结束
                s=(LNode*)malloc(sizeof(LNode));          //创建新结点
                s->data=x;
                s->next=L->next;
                L->next=s;                                       //将新结点插入表中，L为头指针
                scanf("%d",&x);                          //读入下一个结点值
        }
        return  L;
}
```

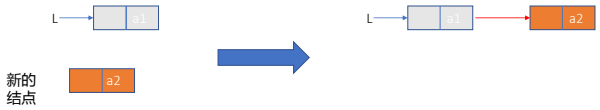


头插法建立单链表，读入数据的顺序与生成的链表中元素的顺序是相反的

2.尾插法建立单链表：

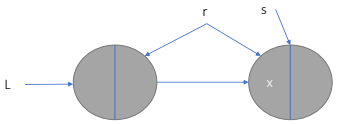
建立新的结点分配内存空间，将新结点插入到当前链表的表尾

需要增加一个指向表尾元素的尾指针



链式存储

```
LinkList CreatList2(LinkList &L){
        int x;
        L=(LinkList)malloc(sizeof(LNode));
        LNode *s, *r=L;                          //r为表尾指针 指向表尾
        scanf("%d",&x);                          //输入结点的值
        while(x!=9999){                          //输入9999表示结束
                s=(LNode *)malloc(sizeof(LNode));
                s->data=x;
                r->next=s;
                r=s;                             //r指向新的表尾结点
                scanf("%d",&x);
        }
        r->next=NULL;                            //尾结点指针置空
        return L;
}
```



尾插法建立单链表，读入数据的顺序与生成的链表中元素的顺序是**相同**的

本节内容

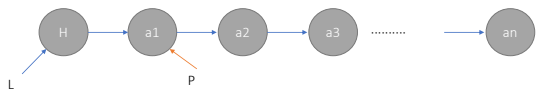
单链表的操作(二)

链式存储

3．按序号查找结点

在单链表中从第一个结点出发，顺指针next域逐个往下搜索，直到找到第i个结点为止,否则返回最后一个结点指针域NULL。

```
LNode * GetElem(LinkList L,int i){
        int j=1;                                 //计数，初始为1
        LNode *p=L->next;                        //第一个结点指针赋给p
        if(i==0) return L;                       //若i等于0，则返回头结点
        if(i<1)  return  NULL;                   //若i无效，则返回NULL
        while(p&&j<i){                           //从第1个结点开始找，查找第i个结点
                p=p->next;
                j++;
        }
        return  p;                               //返回第i个结点的指针，如果i大于表长，直接返回p即可
}
```

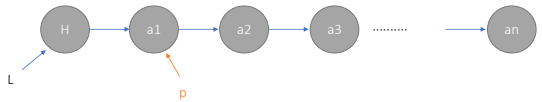


链式存储

4．按值查找结点

从单链表第一个结点开始，由前往后依次比较表中各结点数据域的值，若某结点数据域的值等于给定值e，则返回该结点的指针；若整个单链表中没有这样的结点，则返回NULL。

```
LNode *LocateElem(LinkList L,ElemType e){
        LNode *p=L->next;
        while(p!=NULL&&p->data!=e)//从第1个结点开始查找data域为e的结点
                p=p->next;
        return  p;                               //找到后返回该结点指针，否则返回NULL
}
```



链式存储

5．插入

插入操作是将值为x的新结点插入到单链表的第i个位置上。先检查插入位置的合法性，然后找到待插入位置的前驱结点，即第i−1个结点，再在其后插入新结点。

算法思路：

1.取指向插入位置的前驱结点的指针

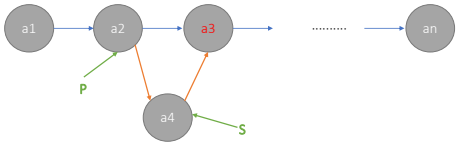
① p=GetElem(L,i-1);

2.令新结点*s的指针域指向*p的后继结点

② s->next=p->next;

3.令结点*p的指针域指向新插入的结点*s

③ p->next=s;



链式存储

6．删除

删除操作是将单链表的第i个结点删除。先检查删除位置的合法性，然后查找表中第i−1个结点，即被删结点的前驱结点，再将其删除。

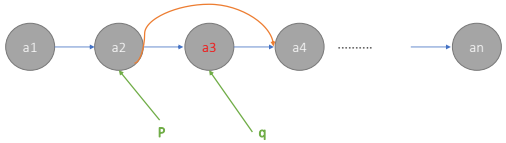
算法思路：

1.取指向删除位置的前驱结点的指针 p=GetElem(L,i-1);

2.取指向删除位置的指针 q=p->next;

3.p指向结点的后继指向被删除结点的后继 p->next=q->next

4.释放删除结点 free(q);



本节内容

双链表

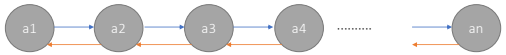
双链表

单链表：单个指针，单向火车

双链表：双指针，电梯

双链表

我们之前学习过单链表，它是酱紫



我们之前学习过单链表，它是酱紫　　双链表

```
typedef struct LNode{          //定义单链表结点类型
   ElemType data;              //数据域
   struct LNode *next;         //指针域
}LNode, *LinkList;
```

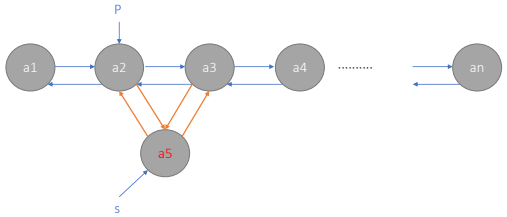
```
typedef struct DNode{          //定义单链表结点类型
   ElemType data;              //数据域
   struct DNode *prior,*next;  //前驱和后继指针
}DNode, *DLinkList;
```


双链表

双链表的操作：插入 删除

1.插入：(方法不唯一)

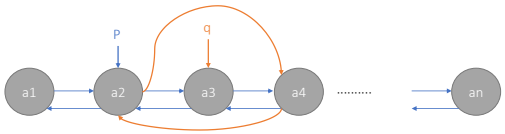
① s->next=p->next;
② p->next->prior=s;
③ s->prior=p;
④ p->next=s;



双链表

2.删除：

① p->next=q->next;
② q->next->prior=p;
③ free(q);



本节内容

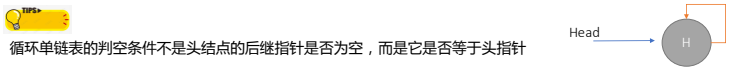
循环链表 && 静态链表

循环单链表

循环单链表：循环单链表和单链表的区别在于，表中最后一个结点的指针不是NULL，而改为指向头结点，从而整个链表形成一个环

单链表

循环单链表



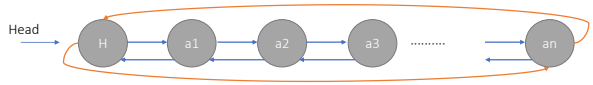
循环单链表的判空条件不是头结点的后继指针是否为空，而是它是否等于头指针

在单链表中访问最后一个结点需要O(n)时间，那有办法在O(1)的时间由链表指针访问到最后一个结点吗？

循环双链表

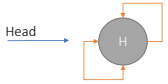
循环双链表：类比循环单链表，循环双链表链表区别于双链表就是首尾结点构成环

双链表:



循环双链表

当循环双链表为空表时，其头结点的prior域和next域都等于Head。



静态链表

前面我们学习的链表大多是指针来实现的，对于一些语言，如Basic，由于没有指针，那链表结构是不是就无法实现了呢？

静态链表：静态链表是用数组来描述线性表的链式存储结构。

静态链表仍然包含数据域和指针域（数组下标），又称游标。

```
#define MaxSize 50                  //静态链表的最大长度
typedef int ElemType                    //静态链表的数据类型假定为int
typedef struct{                         //静态链表结构类型的定义
        ElemType data;                  //数据域：存储数据元素
        int next;                       //指针域：下一个元素的数组下标
} SLinkList[MaxSize];
```


静态链表

数组第一个元素不存储数据，它的指针域存储第一个元素所在的数组下标。
链表最后一个元素的指针域值为-1。

	数据域data	指针域next
0		2
1	b	6
2	a	1
3	d	-1
4		
5		
6	c	3

静态链表

静态链表对应的单链表